Case-Base Maintenance for CCBR-Based Process Evolution

[1] Quality Engineering Research Group, University of Innsbruck, Austria
Barbara.Weber@uibk.ac.at
[2] Information Systems Group, University of Twente, The Netherlands
m.u.reichert@utwente.nl
[3] Evolution Consulting, Innsbruck, Austria
werner.wild@evolution.at

Abstract. The success of a company more and more depends on its ability to flexibly and quickly react to changes. Combining process management techniques and conversational case-based reasoning (CCBR) allows for flexibly aligning the business processes to new requirements by providing integrated process life cycle support. This includes the adaptation of business processes to changing needs by allowing deviations from the predefined process model, the memorization and the reuse of these deviations using CCBR, and the derivation of process improvements from cases. However, to effectively support users during the whole process life cycle, the quality of the data maintained in the case base (CB) is essential. Low problem solving efficiency of the CCBR system as well as inconsistent or inaccurate cases can limit user acceptance. In this paper we describe fundamental requirements for CB maintenance, which arise when integrating business process management (BPM) and CCBR and elaborate our approach to meeting these requirements.

1 Introduction

The economic success of an enterprise more and more depends on its ability to flexibly align its business processes to quickly react to changes, e.g., in the market or in technology requiring flexible "process-aware" information systems (PAIS) [1] to effectively support this alignment [2,3]. Authorized users must be allowed to deviate from the pre-defined process model to deal with unanticipated situations. For example, in a specific patient treatment process the patient's current medication may have to be changed due to an allergic reaction, i.e., the process instance representing this treatment procedure may have to be dynamically adapted (e.g., by deleting, adding or moving process activities). In addition to such instance-specific changes, PAIS must be able to adapt to changes of the underlying business processes themselves, e.g., due to reengineering efforts [4] or the introduction of new laws. For instance, it might become necessary to inform not only newly admitted patients about the risks of a medical treatment, but also patients with an ongoing treatment process who have not obtained their medication yet.

T.R. Roth-Berghofer et al. (Eds.): ECCBR 2006, LNAI 4106, pp. 106–120, 2006.

The need for more flexible PAIS has been recognized for several years [2,3]. Existing technology supports ad-hoc changes at the process instance level (i.e., run time adaptations of individual process instances) as well as changes at the process type level (i.e., changes of a process model) [5]. In CBRFlow [6], for example, we have applied conversational case-based reasoning (CCBR) to assist users in defining ad-hoc changes and in capturing contextual knowledge about these changes; furthermore, CBRFlow supports the reuse of information about ad-hoc changes when defining new ones. CCBR is an extension of the CBR paradigm, which actively involves users in the inference process [7]. A CCBR system can be characterized as an interactive system that, via a mixed-initiative dialogue, guides users through a question-answering sequence in a case retrieval context (cf. Fig 3). In [8,9] we have extended our approach to a complete framework for integrated process life cycle support as knowledge from the case base (CB) is applied to continuously derive improved process models.

To provide adequate process life cycle support, the quality of the data maintained in the CB is essential. For example, the presence of inconsistent or inaccurate cases in the CB is likely to reduce problem-solving efficiency and solution quality and limit user acceptance. The need for CB maintenance arises as cases covering ad-hoc deviations are added by users and not by experienced process engineers and the CB incrementally evolves over time. New cases are added in exceptional situations which have never been dealt with before. To ensure accuracy of the cases and to improve the performance of the CB, CB maintenance becomes crucial when the CB grows. Due to environmental changes and process evolution updates of the CB itself become necessary. Potential process improvements are suggested by the CCBR system, leading to changes in the process model. To maintain consistency of the cases in the CB and to avoid redundancies between the updated process model and the CB, cases leading to or affected by these updates must be revised or possibly removed from the CB version. The process engineer must be supported by suitable maintenance policies and tools.

In our previous work we focused on the integration of business process management (BPM) and CCBR. We developed detailed concepts for memorization and reuse of process instance changes, which allow to derive process (model) improvements from cases [6,9,10]. So far, CB maintenance issues have not been considered in detail, but are a logical next step to provide comprehensive support for process life cycle management. Section 2 introduces basic concepts related to process life cycle support. Section 3 discusses fundamental requirements for CB maintenance in the BPM domain. How we meet these requirements in our approach is described in Section 4. Section 5 discusses related work. We conclude with a summary and an outlook in Section 6.

2 Integrated Process Life Cycle Support Through CCBR

2.1 Business Process Management Fundamentals

PAIS enable users to model, execute, and monitor a company's business processes. In general, orchestration of a business process is based on a predefined process

model, called a *process schema*, consisting of the tasks to be executed (i.e., activities), their dependencies (e.g., control and data flow), organizational entities performing these tasks (i.e., actors) and business objects which provide data for the activities. Each business case is handled by a newly created *process instance* and executed as specified in the underlying process schema.

For each business process (e.g., booking a business trip or handling an order) a *process type* T has to be defined. One or more *process schemes* may exist reflecting different versions of T. In Fig. 1, for example, process schemes S and S' correspond to different versions of the same process type. Based on a process schema new *process instances* $I_1, \ldots, I_m$ can be created and executed.

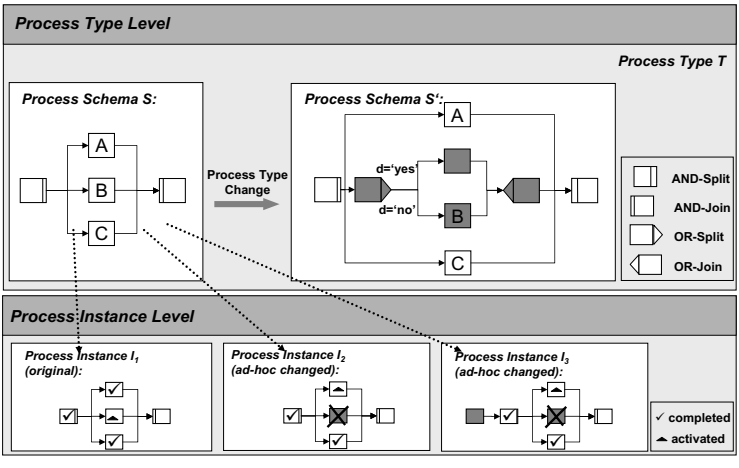

Fig. 1. Different Levels of Process Change

As motivated above PAIS must support process type as well as process instance changes. Changes to a process type T that are necessary to cover the evolution of real-world business processes are performed by the process engineer [5,11,12]. As a result we obtain a new schema version S' of the same type T (cf. Fig. 1) and the execution of future process instances is then based on S'. In contrast, ad-hoc changes of individual process instances are performed by process participants (i.e., end users). Such changes become necessary to react to exceptional situations [2,6,13]. The effects of such instance-specific changes are kept local to the respective process instance, i.e., they do not affect other process instances of the same type. In Fig. 1 instance I_2 has been individually modified by dynamically deleting activity B. Thus the respective execution schema of I_2 deviates from the original process schema S this instance was derived from.

2.2 Integrated Process Life Cycle Support - Overview

Fig. 2 shows how integrated process life cycle support can be achieved by combining BPM technology and CCBR. At build time an initial representation of

a business process is created either by process analysis or by process mining (i.e., by observing process and task executions) (1). At run time new process instances can then be created from the predefined process schema (2). In general, process instances are executed according to the process schema they were derived from, and activities are assigned to process participants to perform the respective tasks (3). However, when exceptional situations occur at the process instance level, process participants must be able to deviate from the predefined schema. Users can either define an ad-hoc deviation from scratch and document the reasons for the changes in the CB, or they can reuse a previously specified ad-hoc modification from the CB (4). The PAIS monitors how often a particular schema is instantiated and how often deviations occur. When a particular ad-hoc modification is frequently reused, the process engineer is notified that a process type change may have to be performed (5). The process engineer can then evolve the process schema (6). In addition, existing cases which are still relevant for the new process schema version are migrated to a new version of the CB (7).

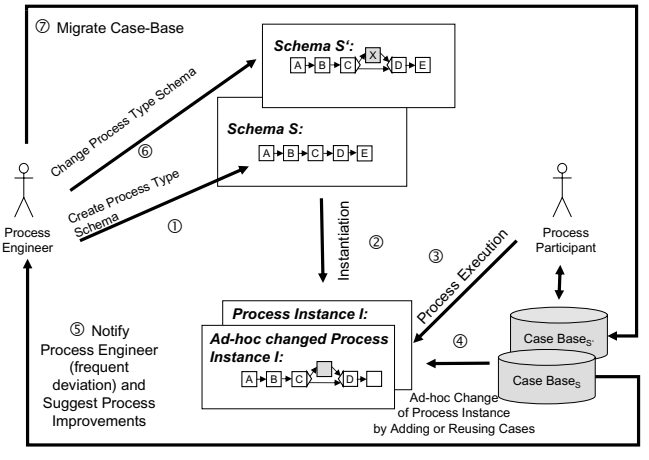

Fig. 2. Integrated Process Life Cycle Support (adapted from [10])

2.3 Case Representation and Reuse

In this section we describe how CCBR is used to capture the semantics of process instance changes, how these changes are memorized, and how they can be retrieved and reused when similar situations occur (for details see [8]).

Case Representation. In our approach a case c represents a concrete ad-hoc modification of one or more process instances. It provides the context of and the reasons for the deviation (cf. Fig. 3). If no similar cases can be found when introducing a process instance change, the user adds a new case with the respective change information to the system. A case consists of a textual problem description *pd* which briefly describes the exceptional situation that led to the

ad-hoc deviation. The reasons for the change are described as question-answer (QA) pairs $\{q_1a_1, \ldots, q_na_n\}$ each of which denotes one particular condition; QA pairs are also used to retrieve cases when similar problems arise in the future. The solution part *sol* (i.e., the action list) contains the applied change operations.

Definition 1 (Case). *A case c is a tuple (pd, qaSet, sol) where*

- *pd is a textual problem description*
- *qaSet* $= \{q_1a_1, \ldots, q_na_n\}$ *denotes a set of question-answer pairs*
- *sol* $= \{ op_j \mid op_j = (opType_j, s_j, paramList_j)$, *j = 1, ..., k*$\}$ *is the solution part of the case denoting a list of change operations (i.e., the changes that have been applied to one or more process instances)*[1].

The question of a QA pair is usually free text, however, to reduce duplicates it can also be selected from a list of already existing questions in that CB. The answer can either be free text or a structured answer expression (cf. Fig 3). Answer expressions allow using contextual information already kept in the PAIS (e.g., due to legal requirements), thus avoiding redundant data entry. Questions with answer expressions can be evaluated automatically by retrieving values for their context attributes from existing data in the system, i.e., they do not have to be answered by users, thus preventing errors and saving time. Free text answers are used when no suitable context attributes are defined within the system or the user is not trained to write answer expressions. For instance, the second QA pair in Fig. 3 contains an answer expression using the context attribute *Patient.age* and can be evaluated automatically. In contrast, the answer in the first QA pair is free text provided by the user.

All information on process instance changes related to a process schema version S is stored as cases in the associated CB of S.

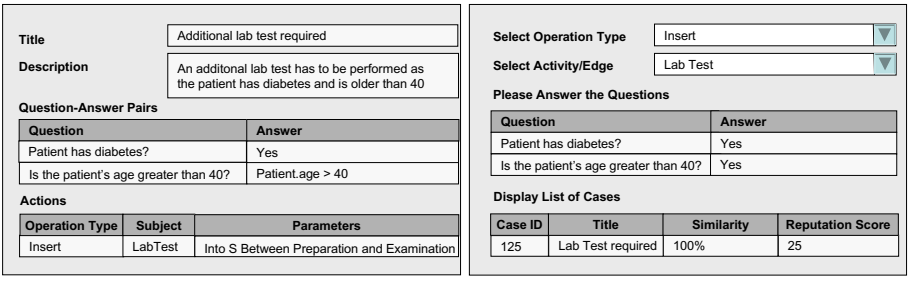

Fig. 3. Sample CCBR Dialogs - Adding a New Case and Retrieving Similar Cases

Definition 2 (Case Base). *A case base* CB_S *is a tuple (S,* $\{c_1, \ldots, c_m\}$, $freq_S$*) where*

- *S denotes the schema version the case base is related to*

[1] An operation $op_j := (opType_j, s_j, paramList_j)$ (j = 1, ..., m) consists of operation type opType_j, subject s_j of the change, and parameter list paramList_j.

- $\{c_1, \ldots, c_m\}$ *denotes a set of cases (cf. Def. 1)*
- $freq_S(c_i) \in \mathbb{N}$ *denotes the frequency with which case* c_i *has been (re–)used in connection with schema version S, formally:* $freq_S$*:* $\{c_1, \ldots, c_m\} \mapsto \mathbb{N}$

Case Retrieval. When deviations from the predefined process schema become necessary the user initiates a case retrieval dialogue in the CCBR component (cf. Fig 3). The system then assists her in finding already stored similar cases (i.e., change scenarios in our context) by presenting a set of questions. Questions with an answer expression are evaluated by automatically retrieving the values of the context attributes. Based on this the system then searches for similar cases by calculating the similarity for each case in the CB and it displays the top n ranked cases (ordered by decreasing similarity) with their reputation score (for details see Section 4.1). Similarity is calculated by dividing the number of correctly answered questions minus the number of incorrectly answered questions by the total number of questions in the case. The user then has different options:

1. The user can directly answer any of the remaining unanswered questions (in arbitrary order), similarity is then recalculated and the n most similar cases are displayed to the user.
2. The user can apply a filter to the case-base (e.g., by only considering cases whose solution part contains a particular change operation). Then all cases not matching the filter criteria are removed from the displayed list of cases.
3. The user can decide to review one of the displayed cases. The case description is then shown to the user.
4. The user can select one of the displayed cases for reuse. The actions specified in the solution part of the case are then forwarded to and carried out by the PAIS. The reuse counter of the case is incremented.

3 Requirements for CB Maintenance

In this section we derive fundamental requirements for CB maintenance in the described scenario. The requirements are aligned with the three top-level performance objectives for CBR systems (cf. [14]): problem-solving efficiency (i.e., average problem solving time), competence (i.e., range of target problems solved) and solution quality (i.e., average quality of a proposed solution).

Req. 1 (Accuracy of the Cases): When using CCBR for memorization and reuse of ad-hoc modifications the CB incrementally evolves over time as new cases are added by end users when exceptions occur. Our approach already guarantees syntactical correctness of the solution part, i.e., the application of change operations to a process schema always results in a syntactically correct process schema [2]. However, semantical correctness of cases must be ensured as well. When cases are added to the CB by inexperienced users it can not always be prevented that inaccurate or low quality cases are added to the CB; however, it must at least be ensured that incorrect cases will not be reused.

Req. 2 (Refactoring QA Pairs): Whenever possible, answer expressions which can be automatically evaluated should be used instead of free text to ease the retrieval process and to increase problem solving efficiency. However, in practice free text QA pairs are entered for good reasons, e.g., the user is unaware of relevant context attributes, she is not trained to formulate answer expressions, or there are no suitable context attributes available in the system when entering the case. The process engineer should be supported in all of the scenarios described above to refactor free text to answer expressions later on.

Req. 3 (Detecting and Handling Inter-Case Dependencies): Occasionally, more than one case may have been applied to a particular process instance. Such dependencies between cases can be observed by analyzing log data (e.g., whenever case c_1 has been applied to a process instance, case c_2 has been applied to this instance as well, i.e., inter-case dependencies exist). When two cases are only used in combination they should be merged to increase problem solving efficiency. When two cases are not always used in combination, but their co-occurrence is frequent, the system should remind users to consider applying the dependent case(s) as well (e.g., by displaying dependent cases).

Req. 4 (Support for CB Migration): Even if cases have been accurate when they were added to the CB, they can become outdated over time. For instance, the evolution of a process schema S (i.e., continuous adaptation schema S to organizational and environmental changes) may reduce the accuracy of parts of the schema-specific CB. After a process type change a subset of the knowledge encoded in the cases may now be captured in the new process schema version S'. The challenge is to migrate only those cases to the new CB version which are still relevant. Cases affected by the process change must be revised by the process engineer or removed from the CB if they are no longer needed.

Additional Requirements. When a CB evolves iteratively, the risk of inconsistencies due to duplicate cases increases and should be mitigated. *Duplicate* cases are either identical, or have the same semantics but are expressed differently. In addition, QA pairs with the same semantics, but different wording, should be avoided, e.g., when entering a new case the user should be supported to reuse already existing QA pairs.

4 Approach to CB Maintenance

In this section we present our approach to CB maintenance and describe how we address the requirements from Section 3.

4.1 Accuracy of the Cases

The accuracy of the cases maintained within a CB is crucial for the overall performance of the CBR system and consequently for the trust users have in it. Particularly, if cases are added by end users adequate evaluation mechanisms for ensuring quality become essential. Like Cheetham and Price [15] we propose to augment the CBR cycle with the ability to determine the confidence users have in the accuracy of individual solutions. In [8], we use the concept of reputation to indicate

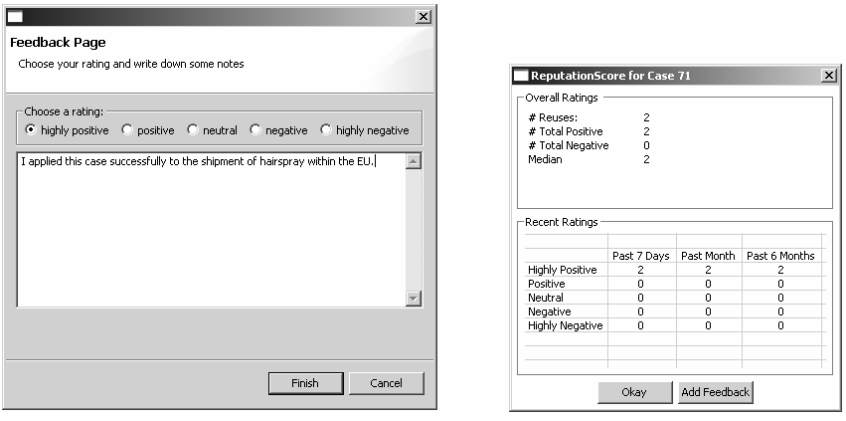

Fig. 4. Feedback forms

how successfully an ad-hoc modification – represented by a case – was reused in the past, i.e., to which degree that case has contributed to the performance of the CB, thus indicating the confidence in the accuracy of this case.

Whenever a user adds or reuses a case she is encouraged to provide feedback on the performed process instance change. She can rate the performance of the respective ad-hoc modification with feedback scores 2 (highly positive), 1 (positive), 0 (neutral), -1 (negative), or -2 (highly negative); additional comments can be entered optionally (cf. Fig. 4); the reputation score of a case is then calculated as the sum of feedback scores. While a high reputation score of a case is an indicator of its semantic correctness, negative feedback probably results from problems after performing a process instance change. Negative feedback therefore results in an immediate notification of the process engineer, who may deactivate the case to prevent its further reuse. The case itself, however, remains in the system to allow for learning from failures as well as to maintain traceability.

During case retrieval the CCBR system displays the overall reputation score (cf. Fig. 3) and the ratings for the past 7 days, the past month, and the past 6 months are also available to the user (cf. Fig. 4). Upon request the user can read all comments provided in the past and decide whether the reputation of the case is high enough for her to have confidence in its accuracy.

4.2 Refactoring QA Pairs

As mentioned cases are used to support memorization and reuse of ad-hoc deviations, whereas QA pairs describe the reasons for the deviation. A question is always free text, an answer can be free text or a structured answer expression (cf. Section 2.3). Whenever possible, answer expressions should be used instead of free text to increase problem solving efficiency. While answer expressions can be automatically evaluated by the system (i.e., answer values are automatically inferred from existing data), free text answers have to be provided by the user. However, in practice it is not always feasible to use answer expressions instead of

free text. In the following we describe three scenarios where free text QA pairs are entered into the system, and we sketch maintenance policies for refactoring free text answers to formal answer expressions.

Scenario 1: The end user applies CCBR to handle an exception, but is not knowledgeable enough to specify formal answer expressions. As the exceptional situation has to be resolved quickly, the user enters free text QA pairs to capture the reasons for the deviation and applies the case immediately. In order to increase problem solving efficiency the respective QA pair should be refactored to a formal answer expression later on, if feasible. Thus, whenever the frequency of answering a particular QA pair exceeds a predefined threshold a notification is sent to the process engineer to accomplish this refactoring.

Scenario 2: The end user is unaware of the application context and cannot find suitable context attributes for specifying answer expressions even though they are available in the system. Therefore, the user enters free text to capture the reasons for the deviation. The process engineer is not informed immediately, but only when the respective QA pair has been answered frequently enough, exceeding a threshold value. He can then refactor the free text to an equivalent answer expression to be used during case retrieval instead.

Scenario 3: No suitable context attributes are available within the system to describe the concrete ad-hoc modification. In this scenario, the user must specify the QA pair using free text. As in Scenarios 1 and 2 the process engineer is informed when the QA pair has been answered frequently enough. He can then decide whether to extend the application context and to add the required context attribute. When a new context attribute is inserted into the system, suitable software components (adapters) for retrieving the context attribute values during run time must be provided.

4.3 Detecting and Handling Inter-case Dependencies

Generally, several ad-hoc changes may be applied to a particular process instance over time, and consequently several cases may exist which affect this instance. In Figure 5, case c_1 and c_2 were both applied to process instance I_1. Case c_1 led to the insertion of an additional activity Z between activities B and C, while the application of case c_2 resulted in the deletion of activity D.

Cases applied to the same instance may be independent of each other, or inter-case dependencies may exist. In a medical treatment process, for example, magnet resonance therapy (MRT) must not be performed if the patient has a cardiac pacemaker. However, a different imaging technique like X-ray may be applied instead. As the deletion of the MRT activity triggers the insertion of the X-ray activity, a semantic dependency between these two ad-hoc changes exists. Discovering such inter-case dependencies is crucial to better assist users in defining changes for other instances later on. In order to discover inter-case dependencies we apply process mining techniques and analyze change logs. In our example the change log reveals that cases c_1 and c_2 were not coincidentally applied together to I_1 only, but always appear in combination.

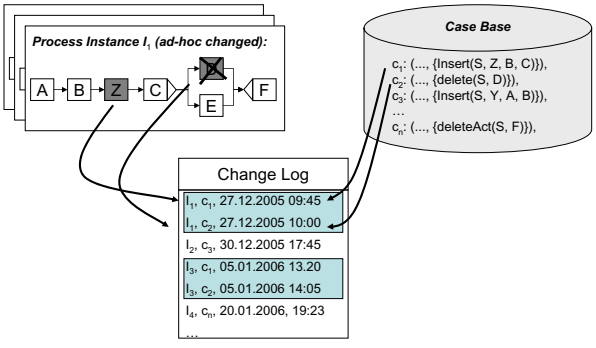

Fig. 5. Discovery of Inter-Case Dependencies

Definition 3 (Strong Inter-Case Dependency). *Let S be a process schema with associated case base CB_S and process instance set $InstanceSet_S$. Further, for case $c \in CB_S$ let $InstanceSet_c \subseteq InstanceSet_S$ denote the set of all process instances to which case c was applied. Then:*

A strong inter-case dependency between $c_1 \in CB_S$ and $c_2 \in CB_S$ exists if $InstanceSet_{c_1} = InstanceSet_{c_2}$, i.e., case c_2 has been applied to all process instances to which c_1 has been applied and vice versa.

If a strong inter-case dependency between c_1 and c_2 exists and the total number of co-occurrences of these two dependent cases exceeds a given threshold n, the process engineer is notified about the option to merge c_1 and c_2. In this situation a new case c' will be created and the original cases c_1 and c_2 be deactivated.[2] The problem description and the QA pairs of c_1 and c_2 are manually merged by the process engineer; the solution parts sol_1 and sol_2, in turn, can be automatically merged by combining the change operations of the original cases in the correct order.

Very often cases co-occur frequently, but do not always co-occur; i.e., there is no strong inter–case dependency between them (cf. Def. 3). In such a scenario the cases cannot be merged. Nevertheless advanced user support can be provided when reusing a case. Assume, for example, that case c_2 has been frequently reused for process instances on the condition that case c_1 has been applied to these instances as well (but not vice versa). When a user applies case c_1 to a process instance and the (conditional) co-occurrence rate $CO(c_2|c_1)$ (see below) exceeds a predefined threshold $m <= 1$, our system will suggest to also consider applying case c_2 to this instance as well.

Definition 4 (Conditional Co-Occurrence Rate). *Let S be a process schema with case base CB_S and let $c_1, c_2 \in CB_S$ be two cases. The conditional co-occurence rate $CO(c_2|c_1)$ denotes the relative frequency of case c_2 on the condition that case c_1 has been applied as well. Formally:*

[2] For traceability reasons respective cases are not deleted, but only deactivated.

$$CO(c_2|c_1) \equiv \frac{|instanceSet_{c_1} \cap instanceSet_{c_2}|}{|instanceSet_{c_1}|}$$

Generally, when reusing a case $c \in CB_S$ at the instance level we present the user all cases $c_k \in CB_S \setminus \{c\}$ with a conditional co-occurence rate $CO(c_k|c)$ exceeding threshold m. The qualified user can then select one or more of the displayed cases and apply them in addition to the previously applied one.

4.4 Support for CB Migration

As discussed in Section 2.1 a PAIS must not only support ad-hoc changes of individual process instances, but also cope with changes at the process type level. An adaptation of process type T may become necessary to react to environmental changes (e.g., the introduction of a new law) or to cover the evolution of business processes. It may also be triggered by the monitoring component of the PAIS, if a particular ad-hoc instance modification has been frequently reused and the process engineer decides to pull this change up to the type level.

Formally, a process type change $\Delta_T = op_1 \ldots op_n$ comprises a sequence of parameterized change operations which are applied to the original type schema S. As a result we obtain a new schema version $S' = S + \Delta_T$ for this type. The challenging questions are how to treat already running process instances of this type and how to evolve case base CB_S.

The execution of future process instances is based on S' whereas already running instances are either continued according to the old schema S or migrated to the new one. Among other constraints the ability to migrate a particular process instance from S to S' depends on its current state; i.e., process instances which have not progressed too far may be migrated to S' and then be executed according to the new schema, whereas instances whose state is not compliant with S' are still executed according to S [5]. On the one hand this enables flexibility when dealing with environmental changes, on the other hand it ensures consistency and correct execution behavior after the change [16,2].

When changing process schema S to $S' = S + \Delta_T$ and migrating selected process instances to S' we must evolve the case base CB_S too. A naive solution would be to ignore all "old" cases for S' (i.e., $CB_{S'} := \emptyset$); another extreme is to associate all existing cases with S' as well (i.e., $CB_{S'} = CB_S$). While the former approach discards all experiences gathered in the past, the latter leads to an inaccurate (i.e., outdated) case base. Note that when applying change $\Delta_T = op_1 \ldots op_n$ to process schema S a subset of the knowledge encoded in the cases from CB_S may then be captured by S'. This particularly holds true if the type change has been triggered by the PAIS itself when the reuse counter of a particular ad-hoc modification (i.e., case) has exceeded a given threshold.

The challenge is to migrate only those cases to $CB_{S'}$ (i.e., to add them to $CB_{S'}$) which remain relevant for future reuse scenarios. This necessitates advanced mechanisms that allow to decide which cases from CB_S can be retained unchanged for $CB_{S'}$, which cases have to be adapted before adding them to $CB_{S'}$, and which cases shall be left out of $CB_{S'}$. In order to answer these questions we have to differentiate whether the process type change triggered by one

or more cases is relevant for all process instances based on S' or only for a particular subset of instances (e.g., an additional activity is only conditionally inserted) [9]. In the following we focus on the former scenario where the solution parts of the triggering cases are directly reflected in the new process schema S' and take a closer look at the relationship between the solution part of a case and the type change Δ_T. Let $\Delta_T = op_1 \ldots op_n$ be a process type change applied to schema S with associated case base CB_S, resulting in the new type schema S'. We consider an arbitrary case $c = (pd_c, qaSet_c, sol_c) \in CB_S$ (with solution part $sol_c = a_1, \ldots, a_k$) and compare it with Δ_T. As the changes are relevant for all instances we can factor out $qaSet_c$ and focus on sol_c only. comparison of parameterized change operations.

- sol_c **and** Δ_T **are equivalent** (i.e., $k = n \wedge a_\nu \equiv op_\nu, \quad \nu = 1 \ldots n$): Cases whose solution part equals Δ_T are not added to $CB_{S'}$. Their "effects" are the same as those of the type change (e.g., case c_1 in Fig 6).
- sol_c **is a subset of** Δ_T (i.e., $\exists\ \mu_1 \ldots \mu_k : 1 \leq \mu_1 < \ldots < \mu_k \leq n : a_\nu \equiv op_{\mu_\nu}, \nu = 1 \ldots k$): Cases whose solution part is a subset of Δ_T are not added to $CB_{S'}$ as their effects are completely covered by the type change (e.g., case c_3 in Fig 6).
- sol_c **and** Δ_T **are disjoint** (i.e., $a_\nu \neq op_\mu, \nu = 1 \ldots k, \mu = 1 \ldots n$): Since the effects of case c are not covered by S' c should be added to $CB_{S'}$. Later reusability requires that actions $a_1, \ldots, a_k$ remain correctly applicable to S'. Note that this might not always be possible due to conflicting change operations. Change Δ_T, for example, might delete an activity from S (e.g., (`delete`, `B`)) to which another operation a from sol_c refers (e.g., $a =$ (`insert`, `X`, `Between A and B`)). We use advanced conflict tests to detect such situations. If no conflicts between sol_c and Δ_T exist, case c can be added to $CB_{S'}$ without further adaptation. Otherwise, the process engineer has to adapt the case in a way that it becomes applicable to S' as well (e.g., by changing the parameterization of actions from sol_c) (e.g., case c_2 in Fig 6).
- sol_c **is a superset of** Δ_T (i.e., $\exists\ \nu_1 \ldots \nu_n : 1 \leq \nu_1 < \ldots < \nu_n \leq k : op_\mu \equiv a_{\nu_\mu}, \mu = 1 \ldots n$): Cases whose solution part is a proper superset of the type change are not directly migrated to $CB_{S'}$. Instead, the process engineer decides whether to add c to $CB_{S'}$ and, if so, how to adapt it. The default adaptation in our system suggests (logically) removing those actions from the solution part of the case whose effects are already captured by S' (i.e., $sol'_c := sol_c \neg \{a_{\nu_1} \ldots a_{\nu_n}\}$) (e.g., case c_4 in Fig 6).
- sol_c **and** Δ_T **are partially overlapping**: Cases in this category are not automatically migrated to $CB_{S'}$. The system supports the process engineer by determining those actions of sol_c whose effects are not reflected by S' (i.e., by calculating the difference set $sol_c \neg \Delta_T$). The process engineer might then decide to migrate case c, after adapting its solution part from sol_c to $sol_c \neg \Delta_T$.

Generally, it is not sufficient to only compare the solution parts of the cases and the process type change. When a process type change triggered by a case is only relevant for a particular subset of process instances (e.g., a lab test should

only be performed for patients older than 40 years suffering from diabetes), we must also look at the corresponding QA pairs and their semantics. Reusing a case at the instance level applies the change operations in its solution part only; the context for this ad-hoc modification is reflected in the case's QA pairs and must be considered by the process engineer when pulling the solution part of the case up to the process type level. Currently we only provide CB migration policies when a process type change is relevant for all process instances. However, we are also investigating migration policies for the scenario just described.

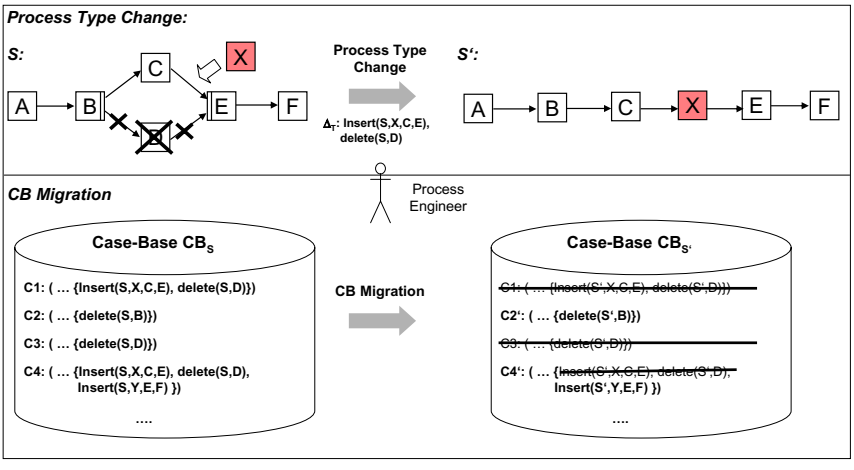

Fig. 6. CB Migration

5 Related Work

Previous research has addressed many aspects of CB maintenance. In general, research on CB maintenance is driven by performance concerns [17] (i.e., problem solving efficiency, CB competence and solution quality of problems solved [18]).

To improve problem solving efficiency while preserving CB competence, strategies for controlling the growth of the CB [19] as well as for selective case retention have been proposed (e.g., [20,21]). In our approach, cases are used for the memorization and the reuse of ad-hoc changes due to exceptions in the business process. In this scenario cases cannot be deleted or only selectively added as traceability of ad-hoc changes must be guaranteed. However, case base migration as proposed by our approach tries to tackle the same problems, aiming to keep the size of the CB compact while preserving competence. When performing CB migration the size of the CB is compressed without reducing the competence of the overall system. Only cases that are still relevant are migrated to the new version of the CB, the removed cases are covered by the new version of the process schema (cf. Section 4.4).

Although a significant body of research exists on CB maintenance none of these approaches deals with inter-case dependencies (i.e., the application of one case triggers the application of another case). In our approach CCBR is not used as a standalone application, but in the broader context of a process management system. This allows us to provide additional context information (e.g., two cases have been applied to the same process instance, i.e., business transaction) facilitating the detection of inter-case dependencies (cf. Section 4.3).

The accuracy of the cases in the case base is crucial for the overall performance of the CB. Therefore Cheetham and Price [15] proposed to augment the CBR cycle with the ability to determine the confidence in the accuracy of individual solutions. However, in our approach accuracy cannot be determined automatically as the semantics of the QA pairs are unknown to the system. Instead, we use the concept of reputation to indicate how successfully a case has been reused in the past, thus indicating the degree of confidence in the accuracy of this case (cf. Section 4.1).

To our knowledge refactoring of free text to answer expressions in a CCBR system has not yet been addressed by existing approaches. They either support structured or unstructured QA pairs, but not both at the same time.

While systematic approaches to CB maintenance like SIAM [22] provide a general framework for building better maintainable CBR systems, this paper focuses on the specifics of the BPM domain.

6 Summary and Outlook

We have derived basic requirements for CB maintenance in the BPM domain (accuracy of cases, refactoring of QA pairs, detecting and handling of inter-case dependencies, and support for CB migration), and we have presented our approach to meeting these requirements. To maintain case quality we apply the concept of reputation score indicating how successfully a case has been applied in the past. Refactoring QA pairs from free text to answer expressions and our approach to dealing with inter-case dependencies contribute to increased problem solving efficiency. Finally, in the context of process evolution we suggest CB migration to deal with outdated cases, keeping the CB compact, while preserving its competence. Ongoing work includes the evaluation of our prototype in a real world scenario. Future work will address the problem of inconsistencies due to redundant cases as we currently only support the reuse of QA pairs by displaying existing ones to the user when adding a new case. We further plan the extension of our CB maintenance approach to also provide policies for CB migration when process type changes are not relevant for all process instances, but only for a particular subset. In this situation the semantics encoded in the QA pairs must be considered when performing a process type change. In summary, CCBR techniques contribute significantly to enriching BPM systems with more semantics and to improving process life cycle support.

References

1. Dumas, M., ter Hofstede, A., van der Aalst, W., eds.: Process Aware Information Systems. Wiley Publishing (2005)
2. Reichert, M., Dadam, P.: ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control. JIIS **10** (1998) 93–129
3. Jørgensen, H.D.: Interactive Process Models. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway (2004)
4. Hammer, M., Stanton, S.: The Reengineering Revolution – The Handbook. Harper Collins Publ. (1995)
5. Rinderle, S., Reichert, M., Dadam, P.: Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. DKE **50** (2004) 9–34
6. Weber, B., Wild, W., Breu, R.: CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In: ECCBR'04, Madrid (2004) 434–448
7. Aha, D.W., Muñoz-Avila, H.: Introduction: Interactive Case-Based Reasoning. Applied Intelligence **14** (2001) 7–8
8. Weber, B., Rinderle, S., Wild, W., Reichert, M.: CCBR–Driven Business Process Evolution. In: ICCBR'05, Chicago (2005) 610–624
9. Rinderle, S., Weber, B., Reichert, M., Wild, W.: Integrating Process Learning and Process Evolution - A Semantics Based Approach. In: BPM 2005. (2005) 252–267
10. Weber, B., Reichert, M., Rinderle, S., Wild, W.: Towards a Framework for the Agile Mining of Business Processes. In: BPM 2005 Workshops. (2005) 191–202
11. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow Evolution. Data and Knowledge Engineering **24** (1998) 211–238
12. Weske, M.: Workflow Management Systems: Formal Foundation, Conceptual Design, Implementation Aspects. Univ. of Münster, Germany (2000) Habil Thesis.
13. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception Handling in Workflow Systems. Applied Intelligence **13** (2000) 125–147
14. Smyth, B., McKenna, E.: Footprint-Based Retrieval. In: ICCBR'99. (1999) 343–357
15. Cheetham, W., Price, J.: Measures of Solution Accuracy in Case-Based Reasoning Systems. In: ECCBR'04. (2004) 106–118
16. Rinderle, S., Reichert, M., Dadam, P.: Flexible Support of Team Processes by Adaptive Workflow Systems. Distributed and Parallel Databases **16** (2004) 91–116
17. Leake, D.B., Wilson, D.C.: Remembering Why to Remember: Performance-Guided Case-Base Maintenance. In: EWCBR'00. (2000) 161–172
18. Smyth, B., McKenna, E.: Footprint-Based Retrieval. Lecture Notes in Computer Science **1650** (1999) 343–357
19. Smyth, B., Keane, M.T.: Remembering to Forget: A Competence-Preserving Case Deletion Policy for Case-Based Reasoning Systems. In: IJCAI'95. (1995) 377–383
20. Smyth, B., McKenna, E.: Building Compact Competent Case-Bases. Lecture Notes in Computer Science **1650** (1999) 329–342
21. Zhu, J., Yang, Q.: Remembering to Add: Competence-preserving Case-Addition Policies for Case Base Maintenance. In: IJCAI'99. (1999) 234–241
22. Roth-Berghofer, T.: Knowledge maintenance of case-based reasoning systems. The SIAM methodology. PhD thesis, University of Kaiserslautern (2002)